**ENHANCED RELIABILITY IN EMBEDDED REAL TIME SYSTEM**

***M.A.Prakruthi1, D.Shalini1, B.Kaleeswari2, S.Dhivya Bharathi2, B.Sree Devi2***

*1B.E Student, 2Assistant Professor*

*1,2 Department of Electronics and Communication Engineering,*

*PERI Institute of Technology*

*kaleeswari.ece.peri@gmail.com*

***Abstract***

***Real-time and embedded computing applications in the first two computing era were rather rare and restricted to a few specialized applications such as space and defense. In the post-PC era of computing, the use of computer systems based on real-time and embedded technologies has already touched every facet of our life and is still growing at a pace that was never seen before. While embedded processing and Internet-enabled devices have now captured everyone’s imagination, these are just a small fraction of applications that have been made possible by real-time systems.***

***If we casually look around us, we can discover many of them often are camouflaged inside simple looking devices. If we observe carefully, we can notice several gadgets and applications which have today become in- dispensable to our everyday life, are in fact based on embedded real-time systems. For example, we have ubiquitous consumer products such as digital cameras, cell phones, microwave ovens, camcorders, video game sets; telecommunication domain products and applications such as set-top boxes, cable modems, voice over IP (VoIP), and video conferencing applications; Office products such as fax machines, laser printers, and security systems. Besides, we encounter real-time systems in hospitals in the form of medical instrumentation equipment’s and imaging systems. There are also a large number of equipment’s and gadgets based on real-time systems which though we normally do not use directly, but nevertheless are still important to our daily life. A few examples of such systems are Internet routers, base stations in cellular systems, industrial plant automation systems, etc.,***

***Key words: Real time System, Embedded System, Reliability, Redundancy, Arduino, SPI Communication***

1. **INTRODUCTION**

A system is called a real-time system, when we need quantitative expression of time (i.e. real-time) to describe the behavior of the system. Commercial usage of computer dates back to a little more than fifty years. This brief period can roughly be divided into mainframe, PC, and post-PC eras of computing. The mainframe era was marked by expensive computers that were quite unaffordable by individuals, and each computer served a large number of users. The PC era saw the emergence of desktops which could be easily be afforded and used by the individual users. The post-PC era is seeing emergence of small and portable computers, and computers embedded in everyday applications, making an individual interact with several computers every day.

Remember that in this definition of a real-time system, it is implicit that all quantitative time measurements are carried out using a physical clock. A chemical plant, whose part behavior description is - when temperature of the reaction chamber attains certain predetermined temperature value, say 250oC, the system automatically switches off the heater within say 30 milliseconds - is clearly a real-time system. Our examples so far were restricted to the description of partial behavior of systems. The complete behavior of a system can be described by listing its response to various external stimuli. It may be noted that all the clauses in the description of the behavior of a real-time system need not involve quantitative measures of time. That is, large parts of a description of the behavior of a system may not have any quantitative expressions of time at all, and still qualify as a real-time system. Any system whose behavior can completely be described without using any quantitative expression of time is of course not a real-time system.

* 1. Objective

In this project, the main objective is to implement a SPI communication between two microcontrollers and to understand the concept on how it works. After the brief literature survey and history and evolution, the architecture with components and its function are explained. Then implementation of spi communication with block diagram and circuit designs are discussed followed by conclusion appended with the program coding.

1. **LITERATURE SURVEY**

Embedded systems design is evolving as businesses are under pressure to innovate faster than ever before. Legacy systems that were once purpose-built must be modernized or give way to new fluid and connected systems. Of course, the need for this transition didn’t happen overnight. Let’s review the history of embedded systems and how things have changed to drive this evolution.

1958 - Whirlwind/SAGE (Semi-Automatic Ground Environment) computer built by MIT and IBM for NORAD is first system to operate in real time. Distributed network of radars for tracking bombers

1960 - SABER (Semi-Automatic Business Environment Research), now SABRE, computer reservations system developed by IBM for American Airlines.

1972 - RDOS (Real-time Disk Operating System) released by Data General for their Nova and Eclipse minicomputers.

1980s - Commercial RTOSes developed: PSOS (Software Components Group), VRTX (Ready

Systems), VxWorks (Wind River Systems), QNX (Quantum Software), LynxOS (LynuxWorks).

1996 - BAG: Real-Time -OS

1998 - SROS: Dynamically Scalable RTOS

1996 - BAG: Real-Time - OS

1998 - SROS: Dynamically Scalable RTOS

1996 - BAG: Real-Time -OS

1998 - SROS: Dynamically Scalable Distributed up RTOS

2002 - TMO-Linux: Linux-based RTOS supporting execution of Time-triggered Message-triggered Objects

2004 - Autonomic Real-Time-OS

1. **ARCHITECTURE**
   1. Components Required

Hardware

In order to build this project, the necessary components are 2Nos. of Arduino UNO R3 board along with LED, Push button, Resistors, Breadboard, Connecting wires etc. Software :

* Arduino IDE 1.8.19
* PC or laptop
  1. Implementation

The steps for implementation of spi connection are as follows:

* First ,make the connections as per circuit diagram (in next chapter)
* check the program code for master as well as slave in the Arduino IDE 1.8.19
* After verification, upload the code into Arduino via USB cable .
* Make sure the components are properly connected to avoid any damages.

1. **SPI COMMUNICATION**
   1. Block Diagram

**SPI (Serial Peripheral Interface)** is a serial communication protocol. SPI interface was found by Motorola in 1970. SPI has a full-duplex connection, which means that the data is sent and received simultaneously. That is a master can send data to a slave and a slave can send data to the master simultaneously. SPI is synchronous serial communication means the clock is required for communication purposes

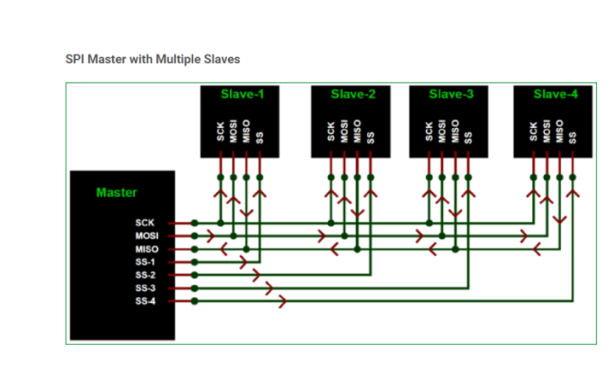


Fig.1: muliple master slave connection

.**SPI has following four lines MISO, MOSI, SS, and CLK**

* **MISO (Master in Slave Out)** - The Slave line for sending data to the master.
* **MOSI (Master Out Slave In)** - The Master line for sending data to the peripherals.
* **SCK (Serial Clock)** - The clock pulses which synchronize data transmission generated by the master.
* **SS (Slave Select)**–Master can use this pin to enable and disable specific devices.
  1. Circuit Diagram

The circuit diagram for the implementation is shown below

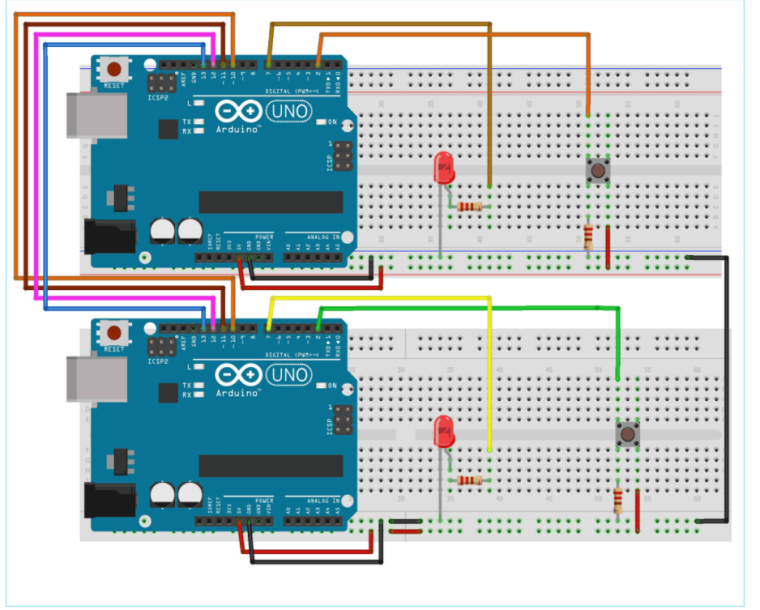


Fig.2: circuit diagram done in Tinkercad

1. **CONCLUSION**

Thus, finally the implementation of spi connection between two Arduino boards has been executed successfully in software mode

**APPENDIX I**

**Program coding:**

***Master Arduino Code:***

//SPI MASTER (ARDUINO)

//SPI COMMUNICATION BETWEEN TWO ARDUINO

#include<SPI.h> //Library for SPI

#define LED 7

#define ipbutton 2

int buttonvalue;

int x;

void setup (void)

{

Serial.begin(115200); //Starts Serial Communication

//at Baud Rate 115200

pinMode(ipbutton,INPUT); //Sets pin 2 as input

pinMode(LED,OUTPUT); //Sets pin 7 as Output

SPI.begin(); //Begins the SPI commnuication

SPI.setClockDivider(SPI\_CLOCK\_DIV8); //Sets clock

//for SPI communication at 8 (16/8=2Mhz)

digitalWrite(SS,HIGH); // Setting SlaveSelect as HIGH

//(So master doesntconnnect with slave)

}

void loop(void)

{

byte Mastersend,Mastereceive;

buttonvalue = digitalRead(ipbutton); //Reads the status

//of the pin 2

if(buttonvalue == HIGH) //Logic for Setting x value

//(To be sent to slave) depending upon input from pin 2

{

x = 1;

}

else

{

x = 0;

}

digitalWrite(SS, LOW); //Starts communication with //Slave connected to master

Mastersend = x;

Mastereceive=SPI.transfer(Mastersend); //Send the

//mastersend value to slave also receives value from slave

if(Mastereceive == 1) //Logic for setting the LED

//output depending upon value received from slave

{

digitalWrite(LED,HIGH); //Sets pin 7 HIGH

//Serial.println("Master LED ON");

}

else

{

digitalWrite(LED,LOW); //Sets pin 7 LOW

Serial.println("Master LED OFF");

}

delay(1000);

}

***Slave Arduino Code:***

//SPI SLAVE (ARDUINO)

//SPI COMMUNICATION BETWEEN TWO ARDUINO

//CIRCUIT DIGEST

//Pramoth.T

#include<SPI.h>

#define LEDpin 7

#define buttonpin 2

volatile boolean received;

volatile byte Slavereceived,Slavesend;

int buttonvalue;

int x;

void setup()

{

Serial.begin(115200);

pinMode(buttonpin,INPUT); // Setting pin 2 as INPUT

pinMode(LEDpin,OUTPUT) // Setting pin 7 as OUTPUT

pinMode(MISO,OUTPUT); //Sets MISO as OUTPUT

//(Have to Send data to Master IN

SPCR |= \_BV(SPE); //Turn on SPI in Slave Mode

received = false;

SPI.attachInterrupt(); //Interuupt ON is set for SPI

//commnucation

}

ISR (SPI\_STC\_vect) //Inerrrput routine function

{

Slavereceived = SPDR; // Value received from master

//if store in variable slave received

received = true; //Sets received as True

}

void loop()

{ if(received) //Logic to SET LED ON OR OFF

//depending upon the value received from master

{

if (Slavereceived==1)

{

digitalWrite(LEDpin,HIGH); //Sets pin 7 as HIGH LED

//ON

Serial.println("Slave LED ON");

}else

{

digitalWrite(LEDpin,LOW); //Sets pin 7 as LOW LED

//OFF

Serial.println("Slave LED OFF");

}

buttonvalue = digitalRead(buttonpin); // Reads the status

//of the pin 2

if (buttonvalue == HIGH) //Logic to set the value

//of x to send to master

{

x=1;

}else

{

x=0;

}

Slavesend=x;

SPDR = Slavesend; //Sends the x value to master

//via SPDR

delay(1000);

}

}

**APPENDIX II**

**OUTPUT IMAGES:**

1. Type the code in arduino:

MASTER:

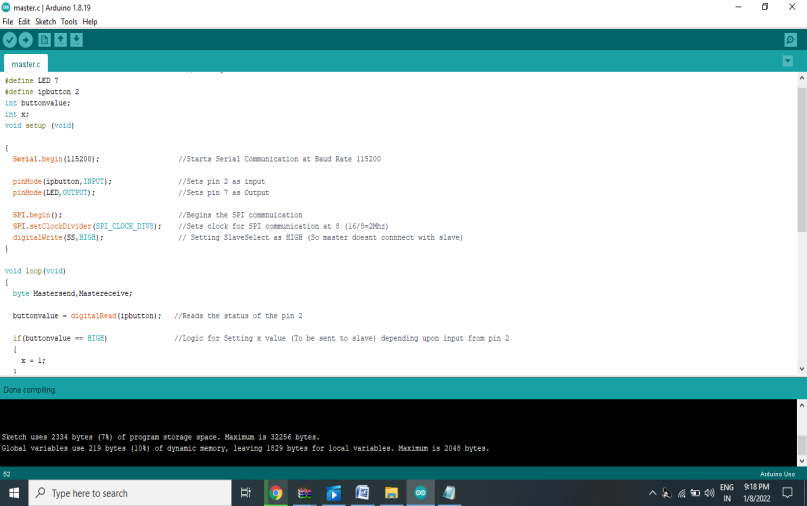


Figure 3. Slave

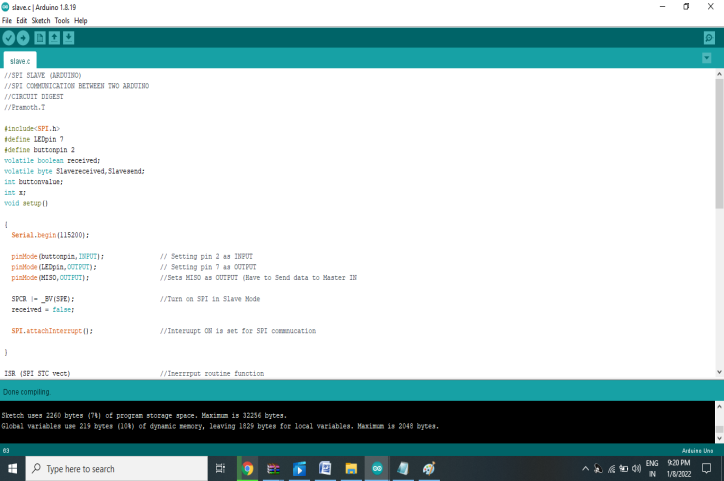


Figure 4.Combining the master and slave code

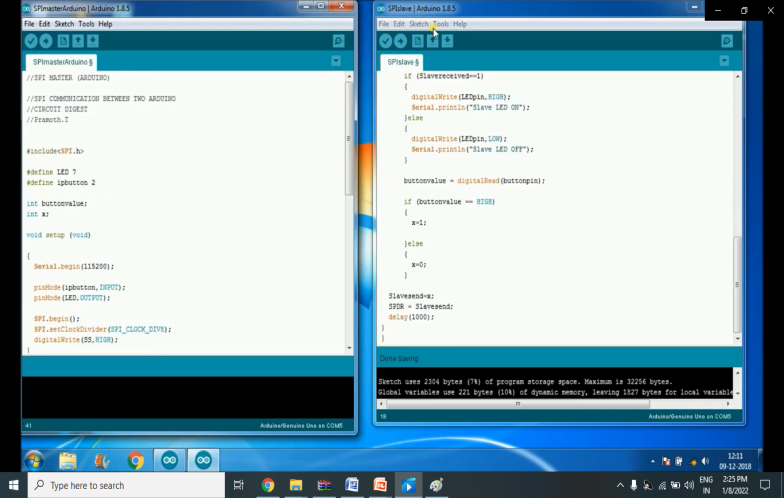


Figure5. Resetting both the Arduinos, by pressing button at same time

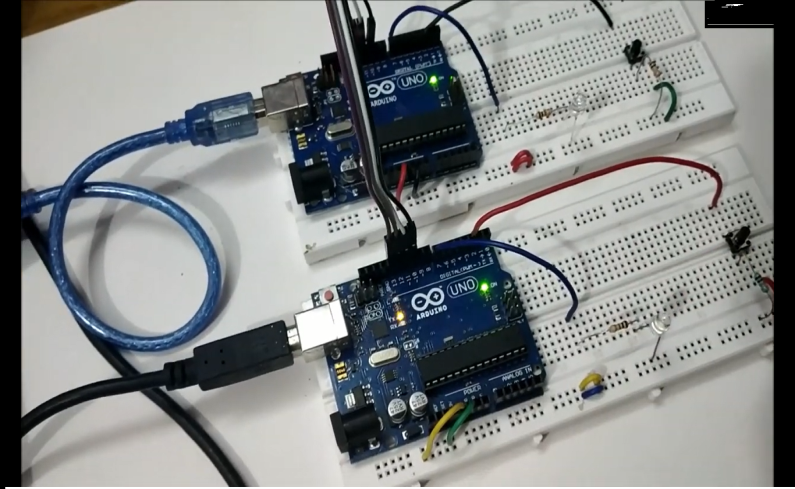


Figure 6. when a push button connected to the slave is pressed, the LED connected to the

Master glows

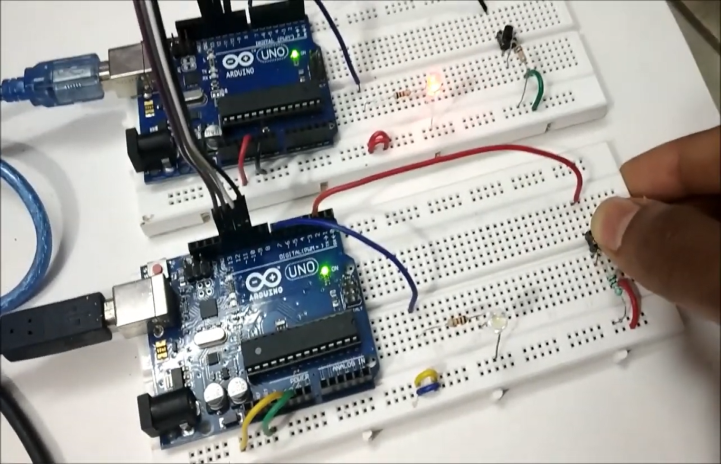


Figure 7. When a push button at master side is pressed,the LED at slave glows

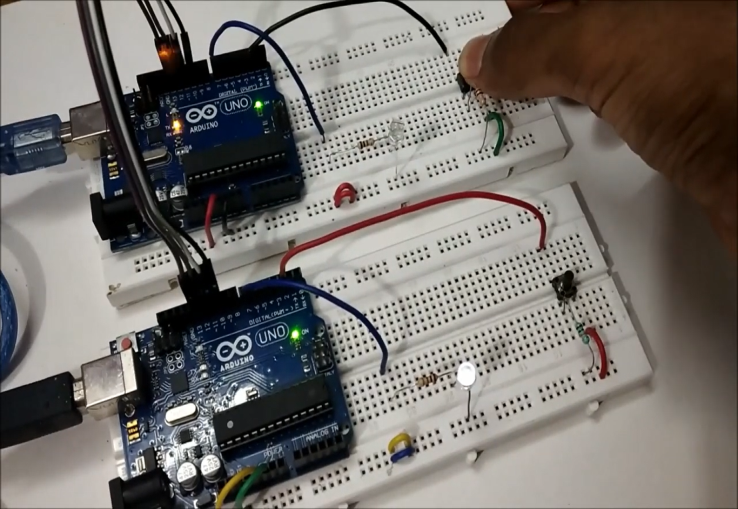


Figure 8.when both buttons are pressed, both the LEDs are turned on.

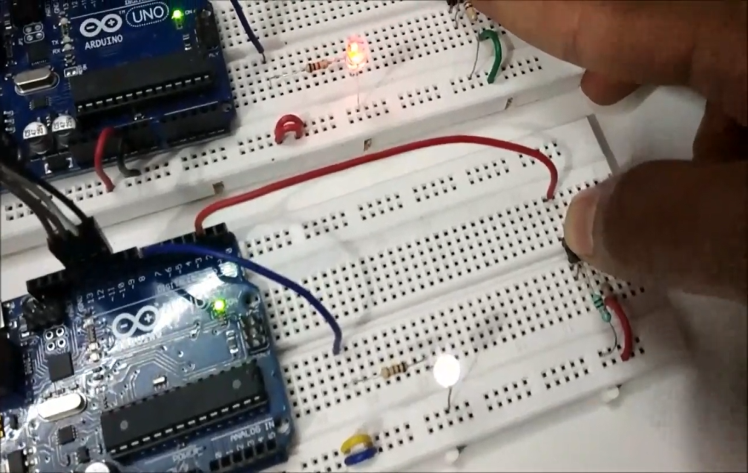
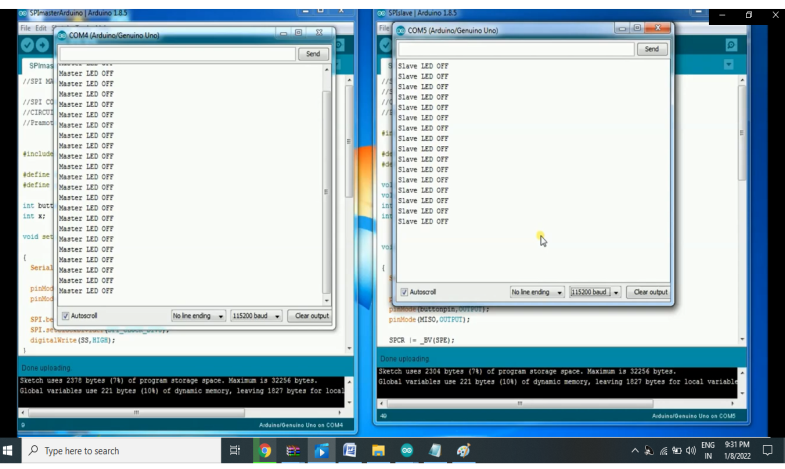


Figure 9. Checking the monitor for the status



**REFERENCES**

1.https://en.m.wikipedia.org/wiki/Embedded\_systems

2.www.circuitdigest.com

3.blogs.winddriver.com

4.www.embedded.com

5.https://youtu.be/wqzNWMJypsw